



Explorador de un lenguaje loco

Lenguaje para discapacitados

IC-5701 Compiladores E Intérpretes

Sánchez Sánchez Miguel David - 2019061555

Sánchez Vargas Kaled - 2019160584

Lumbi Murillo Kimberly Verónica - 2020146765

Vargas Fletes Juan José - 2020035292

Picado Arias Andrey Fabián- 2020135773

TEC Costa Rica, Cartago

Prof. Aurelio Sanabria, grupo 2

II Semestre 2022 – 26 de septiembre

Tabla de contenidos

Demostración y discusión	3
Lecciones aprendidas	4
Pruebas	5
Error de sintaxis #1: Palabras reservadas en mayúsculas	5
Ejecución correcta #1: Fibonacci	6
Error de sintaxis #2: Ejemplo de multiplicación	7
Detección correcta #2: Espaciado extra	8
Error de sintaxis #3: Caracteres especiales	9
Detección correcta #3: Escaped Characters.	10
Error de sintaxis #4: String multilínea	11
Detección correcta #4: Flotantes y enteros.	12
Salud mental/ Amor propio/Diversion/Crecimiento personal	13

Demostración y discusión

Para la demostración de cómo se creó el explorador dividiremos las decisiones en las partes más importantes.

1. Búsqueda de archivo compatible

Para seleccionar un archivo, se decidió que en lugar de que el usuario tuviera que insertar la dirección del archivo debido a que esto se vuelve muy tedioso, se decidió utilizar una función de búsqueda de archivo por medio de interfaz gráfica únicamente para esta función. Solamente se pueden abrir archivos con la extensión .bh del lenguaje Buho.

2. Desfragmentación del archivo

Para desfragmentar el archivo primero se divide por líneas de código y luego estas se dividen en los tokens. Por cada lista de componentes que se itera, ya sean líneas o tokens, se lleva la ubicación por la que se va trabajando para utilizarlas en la identificación de errores.

3. Identificador de componentes

Para identificar los componentes se utilizan las regex y se busca que coincidan con las palabras clave o que sean ignoradas si son comentarios o espacios en blanco.

4. Errores

Tras identificar los componentes se revisa el resultado que devuelve el procesador de la línea y si este es del tipo error se mostrará un mensaje que muestra el componente que es incorrecto, en donde se ubica y una descripción simple del problema. Al procesar las líneas por tokens se utiliza un manejo de errores en el que se salta hasta un punto seguro para evitar la continuación de errores en cascada.

Lecciones aprendidas

La primera lección aprendida, y una de las más valiosas que dejarán huella tanto a lo largo de nuestra carrera universitaria como de la vida profesional, es que, contrario a lo que se cree, la implementación de la lógica de varias personas en la programación puede ser una completa ventaja a la hora tanto de implementar como de solucionar problemas en el código.

Otra lección aprendida por la naturaleza del proyecto, es la importancia de la documentación del código. Sobre todo cuando se trabaja con diferentes personas que tienen lógicas diferentes para implementar la misma función como se mencionó anteriormente, es indispensable que exista una “buena comunicación” entre los colegas que modifican el código. Es inclusive una cuestión de empatía.

Aunque parezca un cuento trillado entre programadores, en el equipo experimentamos por nuestra propia cuenta lo positivo de llegar al acuerdo de dejar la solución de un error para “más tarde” y despejarse, ya que después de un tiempo prolongado frente a la computadora sin obtener respuesta, nuestro compañero Kaled no necesitó más de 20 minutos de descanso para encontrar la razón por la que nuestro programa no estaba funcionando como esperábamos.

Para que lo anterior pueda ser llevado a cabo sin remordimiento, también fue necesario aprender el valor del buen manejo del tiempo, e inclusive de la importancia de establecer fechas de entrega a nivel interno del equipo para poder asegurar una entrega final de calidad y a tiempo, sin sacrificar horas de sueño o de trabajo.

Por otro lado, es esencial reconocer las habilidades de cada miembro del equipo, y tomarlas en cuenta en el momento de la repartición de tareas. Aunque puede que un entorno como el nuestro parezca que todos tengamos las mismas competencias (de programar, por ejemplo), no se debe dejar de lado que como seres humanos integrales tenemos competencias en diferentes áreas. En nuestro caso puntual, una sana convivencia durante las reuniones nos permitió darnos cuenta que nuestro compañero Andrey Picado tiene un gran talento para la locución,

y en esta entrega fue un total acierto que él asumiera la narración del vídeo explicativo.

Pruebas

Error de sintaxis #1: Palabras reservadas en mayúsculas

En este caso se detectan los errores para un mal uso de las palabras reservadas de funciones, en el cual están escritas en mayúsculas.

```
Funcion recibirNombre Inicio_Funcion
    texto nombre tiene ""
    recibir_entrada con_comentario "digite el nombre" guardar_en nombre
    devuelve nombre
Final_Funcion
```

En este caso el explorador logra identificar que están mal escritas, arrojar una lista de componentes con las detecciones y alertando al usuario del error.

```
● juanvargas@MacBook-Air-de-Juan:~/proyecto/compiladores$ python3 buho.py "documentacion/Entregable 2 - Explorador/Tests/error_sintaxis.bh" --error
...Error de escritura de componente detectado:
***** Ubicado en : En la línea número 1. En la columna 7. En el componente Funcion.
...TipoComponente.ERROR <Funcion> en 1:7 mal escrito...
*****
...Error de escritura de componente detectado:
***** Ubicado en : En la línea número 1. En la columna 7. 35. En el componente Funcion. Inicio_Funcio.
...TipoComponente.ERROR <Inicio_Funcio> en 1:35 mal escrito...
*****
...Error de escritura de componente detectado:
***** Ubicado en : En la línea número 1. 2. 3. 4. 5. En la columna 7. 35. 12. En el componente Funcion. Inicio_Funcio. Final_Funcio.
...TipoComponente.ERROR <Final_Funcio> en 5:12 mal escrito...
*****
TipoComponente.ERROR <Funcion> en 1:7
TipoComponente.IDENTIFICADOR <recibirNombre> en 1:21
TipoComponente.ERROR <Inicio_Funcio> en 1:35
TipoComponente.TIPO <texto> en 2:8
TipoComponente.IDENTIFICADOR <nombre> en 2:14
TipoComponente.DECLARACION <tiene> en 2:21
TipoComponente.TEXTO <"> en 2:23
TipoComponente.RECIBIMIENTO <recibir_entrada> en 3:18
TipoComponente.RECIBIMIENTO <con_comentario> en 3:33
TipoComponente.TEXTO <"digite el nombre"> en 3:51
TipoComponente.RECIBIMIENTO <guardar_en> en 3:63
TipoComponente.IDENTIFICADOR <nombre> en 3:69
TipoComponente.PALABRA_CLAVE <devuelve> en 4:10
TipoComponente.IDENTIFICADOR <nombre> en 4:17
TipoComponente.ERROR <Final_Funcio> en 5:12
```

Ejecución correcta #1: Fibonacci

Para el caso de Fibonacci se utiliza el siguiente ejemplo de código, el cual es correcto:

```
// Fibonacci

funcion fibo recibe numerico n inicio_funcion
    si n menor 2 inicio_si
        devuelve n
    final_si
    numerico n1 tiene llamar fibo recibe n menos 1
    numerico n2 tiene llamar fibo recibe n menos 2
    devuelve n1 mas n2
final_funcion
```

Y se obtiene el siguiente resultado, sin mostrar errores:

```
juanvargas@MacBook-Air-de-Juan:~/proyecto_compiladores % python3 buho.py "documentacion/Entregable 2 - Explorador/Tests/fibo.bh" -e
TipoComponente.PALABRA_CLAVE <funcion> en 2:8
TipoComponente.IDENTIFICADOR <fibo> en 2:12
TipoComponente.PALABRA_CLAVE <recibe> en 2:20
TipoComponente.TIPO <numerico> en 2:29
TipoComponente.IDENTIFICADOR <n> en 2:30
TipoComponente.IDENTIFICADOR <inicio_funcion> en 2:45
TipoComponente.CONDICIONAL <si> en 3:5
TipoComponente.IDENTIFICADOR <n> en 3:6
TipoComponente.OPERADOR_LOGICO <menor> en 3:13
TipoComponente.NUMERO <> en 3:14
TipoComponente.CONDICIONAL <inicio_si> en 3:24
TipoComponente.PALABRA_CLAVE <devuelve> en 4:12
TipoComponente.IDENTIFICADOR <n> en 4:14
TipoComponente.CONDICIONAL <final_si> en 5:10
TipoComponente.TIPO <numerico> en 7:11
TipoComponente.IDENTIFICADOR <n1> en 7:13
TipoComponente.DECLARACION <tiene> en 7:20
TipoComponente.IDENTIFICADOR <llamar> en 7:26
TipoComponente.IDENTIFICADOR <fibo> en 7:31
TipoComponente.PALABRA_CLAVE <recibe> en 7:39
TipoComponente.IDENTIFICADOR <n> en 7:40
TipoComponente.OPERADOR_ARITMETICO <menos> en 7:47
TipoComponente.NUMERO <1> en 7:48
TipoComponente.TIPO <numerico> en 8:11
TipoComponente.IDENTIFICADOR <n2> en 8:13
TipoComponente.DECLARACION <tiene> en 8:20
TipoComponente.IDENTIFICADOR <llamar> en 8:26
TipoComponente.IDENTIFICADOR <fibo> en 8:31
TipoComponente.PALABRA_CLAVE <recibe> en 8:39
TipoComponente.IDENTIFICADOR <n> en 8:40
TipoComponente.OPERADOR_ARITMETICO <menos> en 8:47
TipoComponente.NUMERO <2> en 8:48
TipoComponente.PALABRA_CLAVE <devuelve> en 10:10
TipoComponente.IDENTIFICADOR <n1> en 10:13
TipoComponente.OPERADOR_ARITMETICO <mas> en 10:18
TipoComponente.IDENTIFICADOR <n2> en 10:20
TipoComponente.PALABRA_CLAVE <final_funcion> en 11:13
```

Error de sintaxis #2: Ejemplo de multiplicación

Para este caso se va a utilizar este ejemplo en el cual se utilizan símbolos no permitidos para los tipos de datos e identificadores.

```
funcion multiplicacion recibe *Numerico x ^nuMerico z inicio_funcion
devuelve x por &z
final_funcion
```

En el cual logra identificar correctamente los símbolos erróneos e informarlos:

```
● juanvargas@MacBook-Air-de-Juan:~/proyecto_compiladores$ python3 buho.py "documentacion/Entregable 2 - Explorador/Tests/multiplicacion.bh" -e
...Error de escritura de componente detectado:
***** Error encontrado *****
Ubicado en : En la línea número 1. En la columna 39. En el componente *Numerico.
...TipoComponente.ERROR      <*Numerico> en 1:39 mal escrito....
***** Error encontrado *****
Ubicado en : En la línea número 1. En la columna 39. 51. 18. En el componente *Numerico. ^nuMerico.
...TipoComponente.ERROR      <^nuMerico> en 1:51 mal escrito...
***** Error encontrado *****
Ubicado en : En la línea número 1. 2. En la columna 39. 51. 18. En el componente *Numerico. ^nuMerico. &.
...TipoComponente.ERROR      <&> en 2:18 mal escrito...
*****
TipoComponente.PALABRA_CLAVE   <funcion> en 1:8
TipoComponente.IDENTIFICADOR  <multiplicacion> en 1:22
TipoComponente.PALABRA_CLAVE  <recibe> en 1:30
TipoComponente.ERROR          <*Numerico> en 1:39
TipoComponente.IDENTIFICADOR <>> en 1:41
TipoComponente.ERROR          <^nuMerico> en 1:51
TipoComponente.IDENTIFICADOR <>> en 1:53
TipoComponente.PALABRA_CLAVE  <inicio_funcion> en 1:69
TipoComponente.PALABRA_CLAVE  <devuelve> en 2:10
TipoComponente.IDENTIFICADOR <>> en 2:12
TipoComponente.IDENTIFICADOR <>> en 2:17
TipoComponente.OPERADOR_ARITMETICO <por> en 2:18
TipoComponente.ERROR          <&> en 2:18
TipoComponente.PALABRA_CLAVE  <final_funcion> en 3:13
```

Detección correcta #2: Espaciado extra

En este caso el ejemplo trata de verificar que el explorador no se ve afectado si hay casos de espaciados extras e innecesarios.

```
funcion      crear_usuario recibe mapa usuarios texto comentario_id texto
comentario_nombre

inicio_funcion

numerico      id tiene 0

texto nombre tiene ""

recibir_entrada con_comentario      comentario_id guardar_en id
recibir_entrada con_comentario comentario_nombre guardar_en nombre

usuarios en id tiene      nombre

final_funcion
```

El explorador los ignora y genera el siguiente resultado:

```
● juanvargas@MacBook-Air-de-Juan:~/proyecto_compiladores$ python3 buho.py "documentacion/Entregable 2 - Explorador/Tests/espaciado.bh" -e
TipoComponente.PALABRA_CLAVE <funcion> en 1:8
TipoComponente.IDENTIFICADOR <crear_usuario> en 1:27
TipoComponente.PALABRA_CLAVE <recibe> en 1:35
TipoComponente.IDENTIFICADOR <mapa> en 1:39
TipoComponente.IDENTIFICADOR <usuarios> en 1:48
TipoComponente.TIPO <texto> en 1:55
TipoComponente.IDENTIFICADOR <comentario_id> en 1:68
TipoComponente.TIPO <texto> en 1:75
TipoComponente.IDENTIFICADOR <comentario_nombre> en 1:92
TipoComponente.IDENTIFICADOR <inicio_funcion> en 2:14
TipoComponente.TIPO <numerico> en 4:11
TipoComponente.IDENTIFICADOR <id> en 4:18
TipoComponente.DECLARACION <tiene> en 4:25
TipoComponente.NUMERO <0> en 4:26
TipoComponente.TIPO <texto> en 5:8
TipoComponente.IDENTIFICADOR <nombre> en 5:14
TipoComponente.DECLARACION <tiene> en 5:21
<"> en 5:23
TipoComponente.TEXTO <recibir_entrada> en 7:18
TipoComponente.RECIBIMIENTO <con_comentario> en 7:33
TipoComponente.IDENTIFICADOR <comentario_id> en 7:52
TipoComponente.RECIBIMIENTO <guardar_en> en 7:64
TipoComponente.IDENTIFICADOR <id> en 7:66
TipoComponente.RECIBIMIENTO <recibir_entrada> en 8:18
TipoComponente.RECIBIMIENTO <con_comentario> en 8:33
TipoComponente.IDENTIFICADOR <comentario_nombre> en 8:50
TipoComponente.RECIBIMIENTO <guardar_en> en 8:62
TipoComponente.IDENTIFICADOR <nombre> en 8:68
TipoComponente.IDENTIFICADOR <usuarios> en 10:10
TipoComponente.IDENTIFICADOR <en> en 10:13
TipoComponente.IDENTIFICADOR <id> en 10:16
TipoComponente.DECLARACION <tiene> en 10:23
TipoComponente.IDENTIFICADOR <nombre> en 10:35
TipoComponente.PALABRA_CLAVE <final_funcion> en 12:13
```

Error de sintaxis #3: Caracteres especiales

Ahora se va a identificar si puede encontrar errores con los caracteres especiales. En este caso, el error solo debería arrojarse en el identificador e ignorar las tildes dentro del string.

```
texto año_náto tiene "Una variable que había tirada en el río"
```

Lo cual hace correctamente con el siguiente resultado:

```
● juanvargas@MacBook-Air-de-Juan:~/proyecto compiladores % python3 buho.py "documentacion/Entregable 2 - Explorador/Tests/caracteres_especiales.bh" -e
...Error de escritura de componente detectado:
***** Error encontrado *****
Ubicado en : En la línea número 1. En la columna 14. En el componente año_náto.
...TipoComponente.ERROR      <año_náto> en 1:14 mal escrito...
*****
TipoComponente.TIPO          <texto> en 1:6
TipoComponente.IDENTIFICADOR  <>a> en 1:7
TipoComponente.ERROR          <año_náto> en 1:14
TipoComponente.DECLARACION    <tiene> en 1:21
TipoComponente.TEXTO          <"Una variable que había tirada en el río"> en 1:62
```

Detección correcta #3: Escaped Characters.

En este caso se quiere verificar que el programa no tiene problemas detectando escaped characters dentro de un string, con el siguiente ejemplo:

```
texto una_oracion tiene "Una pequeña oración\nDe Dos Lineas\t\nO tres?\n/Si"
```

Lo cual logra hacer correctamente:

```
● juanvargas@MacBook-Air-de-Juan proyectoocompiladores % python3 buho.py "documentacion/Entregable 2 - Explorador/Tests/escape_characters.bh" -e
  TipoComponente.TIPO          <texto> en 1:6
  TipoComponente.IDENTIFICADOR <una_oracion> en 1:17
  TipoComponente.DECLARACION   <tiene> en 1:24
  TipoComponente.TEXTO         <"Una pequeña oración\nDe Dos Lineas\t\nO tres?\n/Si"> en 1:75
○ juanvargas@MacBook-Air-de-Juan proyectoocompiladores %
```

Error de sintaxis #4: String multilínea

Nuestro lenguaje no tiene soporte para strings multilínea dentro del código. En este caso lo probaremos con este ejemplo:

```
numerico un_numerito tiene 432
texto un_textito tiene "
    Esto esta malo
"
```

Lo cual logra detectar correctamente y arrojar errores.

```
● juanvargas@MacBook-Air-de-Juan:~/proyecto_compiladores % python3 buho.py "documentacion/Entregable 2 - Explorador/Tests/error_multilinea.bh" -e
...Error de escritura de componente detectado:
***** Error encontrado *****
Ubicado en : En la linea número 1. 2. En la columna 23. En el componente .
...TipoComponente.ERROR      <> en 2:23 mal escrito...
***** 

...Error de escritura de componente detectado:
***** Error encontrado *****
Ubicado en : En la linea número 1. 2. 3. En la columna 23. 12. En el componente . Esto.
...TipoComponente.ERROR      <Esto> en 3:12 mal escrito...
***** 

...Error de escritura de componente detectado:
***** Error encontrado *****
Ubicado en : En la linea número 1. 2. 3. 4. En la columna 23. 12. 4. En el componente . Esto. .
...TipoComponente.ERROR      <> en 4:4 mal escrito...
***** 

TipoComponente.TIPO          <numerico> en 1:9
TipoComponente.IDENTIFICADOR <un_numerito> en 1:20
TipoComponente.DECLARACION   <tiene> en 1:27
TipoComponente.NUMERO        <432> en 1:30
TipoComponente.TIPO          <texto> en 2:6
TipoComponente.IDENTIFICADOR <un_textito> en 2:16
TipoComponente.DECLARACION   <tiene> en 2:23
TipoComponente.ERROR         <> en 2:23
TipoComponente.ERROR         <Esto> en 3:12
TipoComponente.IDENTIFICADOR <esta> en 3:17
TipoComponente.IDENTIFICADOR <malo> en 3:22
TipoComponente.ERROR         <> en 4:4
```

Detección correcta #4: Flotantes y enteros.

Para este caso, vamos a identificar si el explorador puede identificar correctamente un entero de un flotante. Con este ejemplo:

```
flotante un_flotante_muy_largo tiene 342.121312312312  
numerico un_entero tiene 789
```

Lo cual logra hacer correctamente con el siguiente resultado:

```
● juanvargas@MacBook-Air-de-Juan:~/proyecto_compiladores$ python3 buho.py "documentacion/Entregable_2 - Explorador/Tests/floatantes.bh" -e  
TipoComponente.TIPO <flotante> en 1:9  
TipoComponente.IDENTIFICADOR <un_flotante_muy_largo> en 1:30  
TipoComponente.DECLARACION <tiene> en 1:37  
TipoComponente.FLOTANTE <342.121312312312> en 1:53  
TipoComponente.TIPO <numerico> en 2:9  
TipoComponente.IDENTIFICADOR <un_entero> en 2:18  
TipoComponente.DECLARACION <tiene> en 2:25  
TipoComponente.NUMERO <789> en 2:28
```

Salud mental/ Amor propio/Diversion/Crecimiento personal

Autoevaluación hecha por lxs integrantes del equipo de trabajo. Esto se expresa mediante una colección de 5 memes.

Kaled:

Nuestra gramática
esta bien, solo
habrá que cambiarla un poco



Ok, solo hay que
quitar estos tipos de
datos y sera pan comido



Nuestras expresiones
regulares solo
están un poco mal, no
tardamos en arreglarlas



Solo hay que añadirle
espacios a las
palabras que lo ocupan
y todo funcionara perfecto

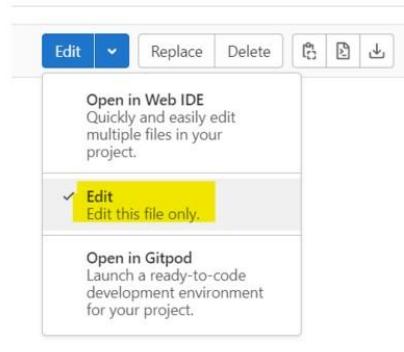


Kimberly:



```
5files — bash — 80x15
apple-MacBook-Pro-4:5files Khamosh$ git push -u origin master
Username for 'https://gitlab.com': firefox008899
Password for 'https://firefox008899@gitlab.com':
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 7.75 MiB | 852.00 KiB/s, done.
Total 7 (delta 0), reused 0 (delta 0)
To https://gitlab.com/firefox008899/test1.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
apple-MacBook-Pro-4:5files Khamosh$
```

Hacer un commit desde la consola



Hacer un commit desde la interfaz de Gitlab

Juan:

**Tu compilador no
compila**



Miguel:

**Viendo como mi
función no está
funcionando y no
sirven los
cambios**

**La función no
se llama**

Andrey:

